Polyglot CerberOS: Resource Security, Interoperability and Multi-Tenancy for IoT Services on a Multilingual Platform

Sven Akkermans
imec-DistriNet, KU Leuven, Celestijnenlaan 200A, Leuven 3001, Belgium
sven.akkermans@cs.kuleuven.be

Bruno Crispo
imec-DistriNet, KU Leuven, Celestijnenlaan 200A, Leuven 3001, Belgium
University of Trento, Via Sommerive 9, I-38123 Trento, Italy
bruno.crispo@cs.kuleuven.be

Wouter Joosen
imec-DistriNet, KU Leuven, Celestijnenlaan 200A, Leuven 3001, Belgium
wouter.joosen@cs.kuleuven.be

Danny Hughes
imec-DistriNet, KU Leuven, Celestijnenlaan 200A, Leuven 3001, Belgium
danny.hughes@cs.kuleuven.be

ABSTRACT

The Internet of Things (IoT) promises to tackle a range of environmental challenges and deliver large efficiency gains in industry by embedding computational intelligence, sensing and control in our physical environment. Multiple independent parties are increasingly seeking to leverage shared IoT infrastructure, using a similar model to the cloud, and thus require constrained IoT devices to become microservice-hosting platforms that can securely and concurrently execute their code and interoperate. This vision demands that heterogeneous services, peripherals and platforms are provided with an expanded set of security guarantees to prevent third-party services from hijacking the platform, resource-level access control and accounting, and strong isolation between running processes to prevent unauthorized access to third-party services and data. This paper introduces Polyglot CerberOS, a resource-secure operating system for multi-tenant IoT devices that is realised through a reconfigurable virtual machine which can simultaneously execute interoperable services, written in different languages. We evaluate Polyglot CerberOS on IETF Class-1 devices running both Java and C services. The results show that interoperability and strong security guarantees for multilingual services on multi-tenant commodity IoT devices are feasible, in terms of performance and memory overhead, and transparent for developers.

CCS CONCEPTS

• Security and privacy → Software security engineering; • Software and its engineering → Embedded software; • Computer systems organization → Embedded and cyber-physical systems;

KEYWORDS

Internet of Things, Services, Interoperability, Security

ACM Reference Format:
Sven Akkermans, Bruno Crispo, Wouter Joosen, and Danny Hughes. 2018. Polyglot CerberOS: Resource Security, Interoperability and Multi-Tenancy for IoT Services on a Multilingual Platform . In *EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous '18), November 5–7, 2018, New York, NY, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3286978.3286997

1 INTRODUCTION

The Internet of Things (IoT) aims to reduce costs for business, improve services for society, tackle environmental challenges and foster economic growth by connecting the digital and physical worlds. These different worlds are being connected through the integration of smart networked *Things* to the Internet to generate data and provide services to multiple parties [15]. The IoT market is rapidly embracing more open innovation where knowledge and resources are shared. This demands more collaboration with a wider variety of participants in the IoT. Siloed approaches simply cannot capture the full value potential of networking all types of services and smart devices in the IoT [11, 23].

State-of-the-art IoT research offers poor support for using constrained devices in the role of microservice-hosting platforms or with very different software and hardware modules [19, 21, 26]. Thus, IoT developers still need to develop custom services for specific platforms using dedicated technologies providing little interoperability. Similarly, IoT infrastructure providers cannot share their devices while maintaining full accountability of their resources. In addition, ensuring secure IoT programs requires custom security approaches that are not general across technologies. These issues make the development of secure multi-tenant IoT devices difficult and form a significant roadblock to achieving commercial large-scale IoT deployments.

To overcome this roadblock, we introduce *Polyglot CerberOS*, a resource-secure multilingual Operating System (OS) for executing multi-tenant interoperable services on constrained IoT devices that delivers instruction-level monitoring and fine-grained resource management through a virtual machine (VM) running on the device. Polyglot CerberOS extends our previous work [2] which focused primarily on providing resource security, the accounting and contractual limitation of service resource use, for Java-based IoT services. Polyglot CerberOS provides isolation, resource security and secure interactions between all services on a device, including those provided by the platform, peripherals and third-party sources. To further promote use by multiple parties, Polyglot CerberOS compiles services in different languages to a common intermediate representation (IR) language ready for execution. Finally, a microservice message bus provides interoperability between heterogeneous services. The core idea of the lightweight microservice bus is to enable flexible and transparent interactions using standards-based serialisation and viewing everything, user programs, device's functions and peripherals, as addressable services.

Summarized, the key contributions of Polyglot CerberOS are:

(1) Ensuring memory isolation, resource security and secure interactions between services, peripherals and platforms.
(2) A paradigm and toolchain that can deploy multiple services in different programming languages on the same platform.
(3) Providing secure interoperability between services, peripherals and platforms through a microservice message bus.
(4) Transparency for developers who can use their preferred programming languages and existing libraries while obtaining high-level language benefits and interoperability.

We implemented a prototype of Polyglot CerberOS on two distinct platforms: a representative IoT platform based on the ATmega1284P, an IETF Class-1 device [7], and Linux-based devices such as a Raspberry Pi or server. Polyglot CerberOS is modular and can work stand-alone but also allows the use of discrete assets from other OSs, such as timers, scheduling and networking. Our experimental evaluation proves that the performance overhead and energy impact of running secure interoperable services written in different programming languages on Polyglot CerberOS are low, making it feasible for real-world IoT scenarios with Class-1 devices.

The remainder of this paper is structured as follows: Section 2 gives the background and motivation for microservices on IoT devices, Section 3 discusses related work, Section 4 describes the design and architecture of Polyglot CerberOS, Section 5 explains the implementation, Section 6 presents the evaluation and Section 7 concludes and proposes future work.

2 SECURE MULTI-TENANT IOT PLATFORMS

Contemporary IoT networks are appearing in many different forms and expanding in scale from smart buildings to smart cities. At the same time, the IoT landscape is becoming more difficult to navigate with many disparate technologies and a notable lack of security, interoperability or support for multi-tenancy. This prevents stakeholders from deploying services on shared platforms [19, 21, 26]. As such, there is an unclear Return-on-Investment (RoI) for large-scale IoT deployments, which demand significant investment in infrastructure and staff to deploy, manage and maintain the system.

Supporting secure multi-tenant IoT deployments allows providers to increase their RoI. Running multiple microservices on IoT platforms satisfies more stakeholders and therefore minimizes the costs arising from the deployment, management and maintenance of devices. IoT infrastructure providers can specialize in hardware deployment and lease out resources on underutilized devices to third parties. Software developers can thus focus on their core competencies while still having access to scalable and secure hardware and software solutions. Cloud computing demonstrates the attractiveness of a model where the same platforms are shared between multiple stakeholders. Realizing secure multi-tenant IoT platforms requires support for strong security mechanisms that can isolate concurrent services and perform fine-grained accounting of and control over resource access as well as providing secure interactions between the heterogeneous software and hardware in the IoT.

Resource Constraints. IoT networks consist of many different types of devices from battery-powered IETF Class-1 devices [7] with 10KB RAM and 100KB ROM through gateway-class devices such as Raspberry Pi's to desktop computers. The resource constraints and battery limits of many IoT devices has driven the development of lightweight OSs such as Contiki, TinyOS and RIOT [4, 13, 20]. These OSs are inherently single user and do not provide memory protection or platform resource management. They also require the use of a specific programming language and limit interoperability. In contrast, a microservice-centric IoT OS would execute efficiently on the constraints of Class-1 embedded devices while securing the execution and resource usage of coexisting services and tackling the problem of language heterogeneity and interoperability between heterogeneous systems.

Security for Microservice-Hosting Platforms. Services need to be isolated from each other and provide guarantees on resource access and use, as motivated by [2]. Key resources, such as memory, energy and peripherals, require management so that services run safely and consistently on a node without mutual interference. CerberOS secures services on constrained platform via two mechanisms: (i) explicitly linking services to a *service contract* that describes their allowed capabilities on the device and (ii) instruction-level monitoring that allows fine-grained device resource management through per-service *resource buckets* that track the resource usage and determines the allowed resource access. These mechanisms secure Java services in CerberOS, providing a software security solution for Class-1 IoT devices. Polyglot CerberOS inherits these benefits by expanding upon this previous work.

The Problem of Language Heterogeneity. The IoT landscape features a wide range of communication protocols, programming languages, OSs and middleware [3]. Commonly, platforms require specific methodologies to develop and deploy services and the user needs to adjust accordingly. Different methodologies mandate the use of different programming languages which may not offer the same safety guarantees, interoperability capabilities or programming model. Beyond burdening programmers with the need to learn new languages, language heterogeneity inhibits the shared and transparent reuse of code on multiple platforms. Likewise, developers have no common foundation of security, memory safety or platform independence. This is a major impediment towards the secure, shared and transparent use of services in the IoT.

Interoperability between Heterogeneous Services. A core issue for the transparent use of heterogeneous services is how they should interoperate across platforms and languages. To do so, services in the IoT need to exchange data and interact with each other transparently. This implies a common communication format or interaction pattern such that no party needs to be concerned with the specific implementation of the other. Achieving this for the many heterogeneous systems in the IoT to enable flexible multi-tenant platforms remains an open challenge [28].

3 RELATED WORK

This section provides an overview of research related to security, multilingualism and interoperability.

Security for Heterogeneous Programs. Security for IoT platforms is steadily advancing but often relies on dedicated hardware support or does not consider multi-tenancy or heterogeneous services. We discuss recent work that proposes security solutions. The Security MicroVisor [12] is a middleware which provides memory isolation and custom security operations using software virtualisation and assembly-level code verification. It is a pure software solution that utilizes a layer of indirection and code verification to realise a trusted computing base for constrained IoT devices. This is a lightweight solution with minimal overhead, but, in contrast to Polyglot CerberOS, the Security MicroVisor operates at a lower level and thus lacks the semantic insights to secure heterogeneous services or their resource access. For example, it cannot limit peripheral access to contractually agreed bounds.

A source of inspiration for Polyglot CerberOS is Sulong [24], an interpreter for LLVM-based languages that run on the Java Virtual Machine. Similarly to Polyglot CerberOS, Sulong leverages an efficient virtual machine and provides memory safety for previously unsafe languages such as C. However, Sulong does not consider constrained devices, the IoT environment or other security aspects such as resource guarantees. In contrast, Polyglot CerberOS focuses specifically on the IoT and considers resource security. Sulong is an example of the active research in exploring the potential of interpreters to secure languages.

Multilingual Systems. There is significant interest in systems which can flexibly work with multiple languages to unburden developers from programming constraints. Brunklaus et al. [9] propose the architecture of a virtual machine to execute multiple languages by defining a service with an interface generic enough to be used by many languages. As with Polyglot CerberOS they leverage a Java Virtual Machine, support two programming languages and introduce similar mechanisms such as generic components and interfaces. However, they do not focus on constrained devices, provide interoperability or solve issues beyond multilingual service execution such as controlling their resource access.

Mote Runner [10] is a multilingual virtual machine for constrained devices. They likewise target multiple programming languages and embedded hardware platforms and note the benefits of using high-level programming languages and virtual machines for easing development and code portability. In contrast to Polyglot CerberOS, they only support strictly-typed languages, such as Java and C#, and use their own bytecode as an IR and strictly limit language features such as strings and multi-dimensional arrays.

Interoperability for Constrained Devices. Interoperability is a common issue in any evolving technology landscape and especially so in the IoT with many different parties, products and vendors. LooCI [16] is a component and binding model for WSNs which promotes loose coupling of components using an event bus. Comparable to Polyglot CerberOS, LooCI uses a loosely-coupled globally-typed event bus to implement service bindings in constrained environments. LooCI achieves a good performance and minimal footprint but it does not support multiple languages on the same device, only providing consistent messaging between C and Java services on different platforms, and does not natively monitor or secure their interactions or resource use.

Kolbe et al. [18] also note the interoperability issues in the IoT and propose PROFICIENT, a productivity tool that enables the publication of IoT data and services with minimal effort. Similarly to Polyglot CerberOS, they leverage open standards to support IoT data/service providers in wrapping proprietary interfaces to join an open IoT ecosystem. This is a top-down approach and requires the generation of an IoT gateway agent that can be deployed on a device at the edge of the 'Web of Things' to create a managed interoperable system. In comparison, the bottoms-up approach of Polyglot CerberOS relies on a common message bus on all platforms that uses open standards to let heterogeneous services interoperate.

Gap Analysis. State-of-the-art research offers efficient solutions to specific aspects of secure multi-tenant platforms such as performance [9, 10] or memory safety [12, 24] but all require either custom solutions; specific to certain technologies, neglect resource security, are unsuited to constrained IoT devices and/or limit the possible programming languages or service interoperability. Table 1 compares properties of programs in key IoT languages to programs running in the CerberOS IR. To the best of our knowledge, current research offers no way to transparently provide isolation and resource security for (i) heterogeneous devices, peripherals and services, (ii) services in different programming languages and (iii) securely interoperating these entities.

4 DESIGN

This section describes the design of Polyglot CerberOS and how it tackles the challenges of secure multi-tenancy for the IoT. The approach of Polyglot CerberOS is based on an intermediate language representation (IR) and a Virtual Machine (VM). A VM with IR are the two core elements that together create a secure interoperable multilingual multi-tenant OS. This section explains how security is achieved between heterogeneous interoperating entities, followed by how a multilingual platform can be designed. Finally, we discuss how to interoperate heterogeneous user and device services in an extensible and transparent manner.

4.1 Securing Heterogeneous Interoperating Services

Secure service deployment, isolation and resource guarantees are critical security aspects in the design of a multi-tenant platform for hosted microservices. Extending the previous version of CerberOS [2], the design leverages a VM and IR to perform instruction-level monitoring and control of resource use.

Table 1: Comparison properties of programs in programming languages in the IoT.

	Memory Safety	Resource Security	Multilingual Support	Interpreted	Impact Constrained Platforms
Native C	No	No	No	No	Low
Native Java	Yes	No	Limited (JNI)	Yes	High
CerberOS IR	Yes	Yes	Yes	Yes	Medium

Service loading occurs over a trusted channel: either through a physical flasher or wirelessly over an AES-128-CCM secured network connection. On loading, services are registered with a resource contract, which is also loaded over a secure channel, and the associated buckets. This contract is configured by an authorized party, e.g., the infrastructure owner, during deployment. The contract and what it allows for should be agreed upon beforehand between the device provider and the service deployer. For instance, it can allow a service to access only certain peripherals and limit bandwidth consumption. As such, every service has a linked contract and associated buckets that are monitored at runtime. Standard runtime, resource access and service interactions must be secured so that a service may not access private data from others, nor deplete its resource allocation due to spurious invocations. Interactions may not circumvent the VM and its resource accounting and thus are also monitored at runtime.

Figure 1 shows how Polyglot CerberOS secures both user and system services and contrasts it to resource access in standard systems. The architecture operates through a system of monitored virtual message signatures, named *stubs*, that are subject to access control and monitoring, as explained in the next sections. This prevents attackers from making unmonitored calls to other services or passing dangerous data in messages (e.g., to cause a buffer overflow attack). The stubs are implemented in the Polyglot CerberOS runtime, which provides access to the CerberOS functionalities, and in the language-specific Guest Language Support Package (GLSP), which provide additional support for languages as needed. Service calls are secure since the GLSP maps them to either the default Polyglot CerberOS runtime or to native code stubs, so all calls are stubs which are checked before execution. Service calls for interactions pass messages, which contain data for actions and an address, and are also secured. Since messages are serialised using common standards, messages can be parsed by the bus to ensure they request contractually allowed actions and to account for the service resource usage. Messages that cannot be parsed are blocked. Hence, an attack cannot hide malicious data in messages. For instance, a message that would cause a write to restricted memory is parsed, checked and blocked from execution.

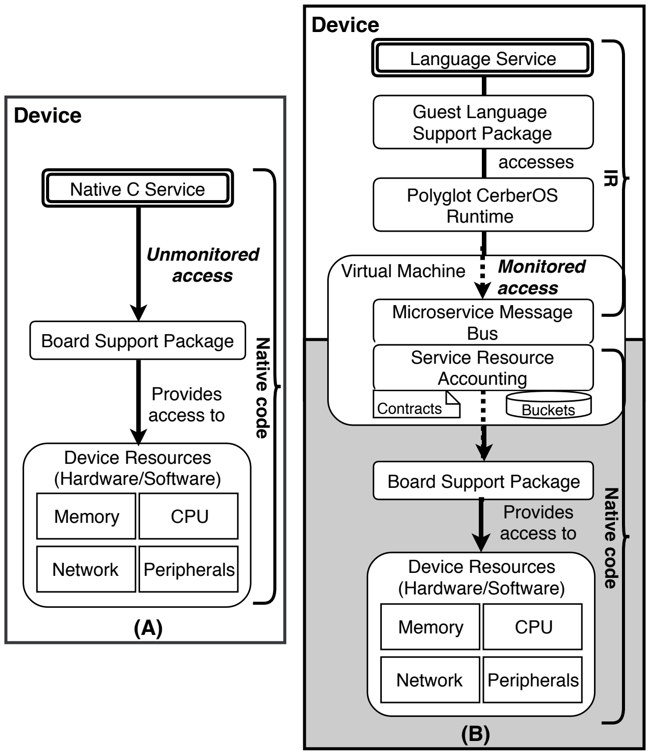


Figure 1: Comparison of how a service accesses device resources for a standard OS (A) and Polyglot CerberOS (B).

4.2 Multilingual Platform Architecture

Requirements on Guest Languages. A general-purpose programming language may vary greatly in syntax, semantics, type system, core libraries and intended application areas [1]. Specifically, languages provide different properties to programmers, e.g., memory safety, execution methodology, runtime overhead, as Table 1 demonstrates. The multilingual platform design of Polyglot CerberOS requires a minimal set of core capabilities of a service in a guest language so that it can be used in a transparent, interoperable IoT application context. The service has to be able to pass data to the Polyglot CerberOS VM using a (i) standard service addressing method and (ii) language-independent serialisation scheme.

Guest Language Platform Independence. Polyglot CerberOS supports languages on different platforms by compiling the language to the IR which it interprets, monitors and executes in native code. The difficulty lies in language dependencies and compiler constraints, which may be unsuitable for constrained platforms. For instance, a C program requires an extensive set of supporting libraries. The design of Polyglot CerberOS introduces a general way of solving these dependencies to achieve platform independence for languages that can be compiled down to the IR.

A system of monitored virtual message signatures, called *stubs*, are subject to access control and monitoring and map to an efficient native implementation to generically solve dependencies [6]. A language executes as normal whether or not the stubbed dependencies in the IR are implemented at the native code level. The design stubs dependencies on the device with the GLSP that is present on the device. These dependencies map to an implementation in native code to perform the desired action.

Toolchain Design. The Polyglot CerberOS toolchain compiles guest languages to the executable IR on the VM. Importantly, the toolchain is transparent and flexible: it starts from a standard IDE which compiles the language to the IR. The design solves required dependencies by providing knowledge of the stubbed dependencies to the toolchain through the GLSP. The service in the language can then be compiled with the stubbed dependencies by the default compilation tool. The burden is not on the programmer, who is agnostic to the compilation, or on the compiler, which is potentially third-party and unmodifiable, but on the toolchain which provides the GLSP which stubs all necessary dependencies, either to existing runtime or in native code. The toolchain links the compiler to the GLSP and ensures the end results is a correctly compiled program in the IR which is linked to the stubbed dependencies. Figure 3 in Section 5 provides a visual example.

4.3 Microservice Message Bus

As argued in Section 2, microservices from multiple parties need to easily interoperate across platform and language barriers to promote their flexible and transparent use. A message bus is a common communication and integration system for connecting multiple components [16, 22]. A bus enables services to interact with each other while maintaining loose coupling.

Messaging Approach. The message bus has four core goals: it should be lightweight, general, transparent for user and device services and message-oriented. The design relies on a minimal set of assumptions on the guest languages and constrained heterogeneous platforms. We assume that all services are able to build a message using a supporting library, i.e., the GLSP, and pass it to the runtime. To enable transparency and interoperability, messages are subject to (i) a common data serialisation scheme and (ii) a standardised addressing scheme. Accordingly, messages include the address of the recipient (i.e., the addressed service) and the serialised data needed to perform the desired operation. The message is passed to the message bus of the platform and interpreted. As discussed in Section 4.1, since messages are serialised using common standards, they can be checked for safety. The bus forwards it to either local services or remote services, depending on the address. For device services, the message is interpreted and the requested action undertaken (e.g., actuation or producing a temperature sensing). For user services, the message is put into a queue and the user service is notified, which can then retrieve and handle the message using a supporting library to deconstruct the message.

Standardised Serialisation and Addressing. In light of the requirements, a lightweight standardised data format is necessary that serialises data to enable useful interactions between services. The design adopts CBOR [8], a binary data serialisation format with small code and message sizes and a focus on extensibility. CBOR is a standard proposed by the IETF and specifically designed for the fast evolving constrained IoT landscape. It is based on the successful JSON data model and hence allows for easy transformation from and to that format by a resource-rich back-end.

Services need to be uniquely addressable to correctly transmit messages to the intended recipients. The design includes the IETF standard COAP [25] which is a specialized web transfer protocol for use with constrained nodes and networks in the IoT. COAP provides an URI scheme for identifying and locating resources; here equivalent to services. Considering device functions, peripherals and user services as COAP resources gives a unique COAP-URI based on the host name, the network (IP) address of the device, and the path of the service. The COAP-URI is used to uniquely address the services present on devices and in the IoT infrastructure. COAP integrates with many data formats, including CBOR and JSON, and is designed with resource discovery and security in mind.

Figure 2 shows how the microservice message bus works. Services view a single unified microservice message bus that operates across different networks, platforms, types of services and languages. Each service has an identifier for addressing, a COAP-URI, and each user service has a service queue for receiving data. User services interact with the message bus and other services through the GLSP, passing CBOR-formatted messages with address information to the bus and retrieving messages from their queue.

5 IMPLEMENTATION

This section details the implementation of the design. At its core, Polyglot CerberOS implements an IR and VM that are extended to create a secure interoperable multi-tenant multilingual OS. Specifically, a micro Java VM serves as the interpreter and Java Bytecode as an IR. Polyglot CerberOS currently supports both services in C and Java. This section goes further into the implementation details important to the challenges outlined in Section 2.

Secure Interoperating Services. Polyglot CerberOS secures services and their interactions by monitoring calls and messages, extending the security mechanisms of [2]. The monitored virtual message signatures are subject to access control and monitoring so that the content of the messages is interpreted and properly accounted and no data or instructions are executed that break resource security guarantees. The VM uses internal data structures to implement *resource contracts* and *buckets*. When user services are loaded, the device initializes these structures based on the contract agreed upon between user and device owner. With each passed message, the service contract and resource buckets are checked for violations based on the content of the message. If the message violates the contract, it is not executed but handled which can be ignoring it, removing the service or changing the message. If the message instruction is to be executed, the resource buckets are adjusted with the amount of consumed resources to support accounting.

Transparent Multilingual Services. The minimal language requirements, defined by Section 4.2, states that services can pass messages that contain an address and CBOR data, which contains the instruction for the recipient. Depending on the language, this can be in the form of function calls with arguments, supported by the GLSP. For instance, a Java service calls a GLSP stub to build a CBOR data array containing a string denoting a command and an integer array serving as arguments to that command. It then passes this message and an address as a string, again using the GLSP, to

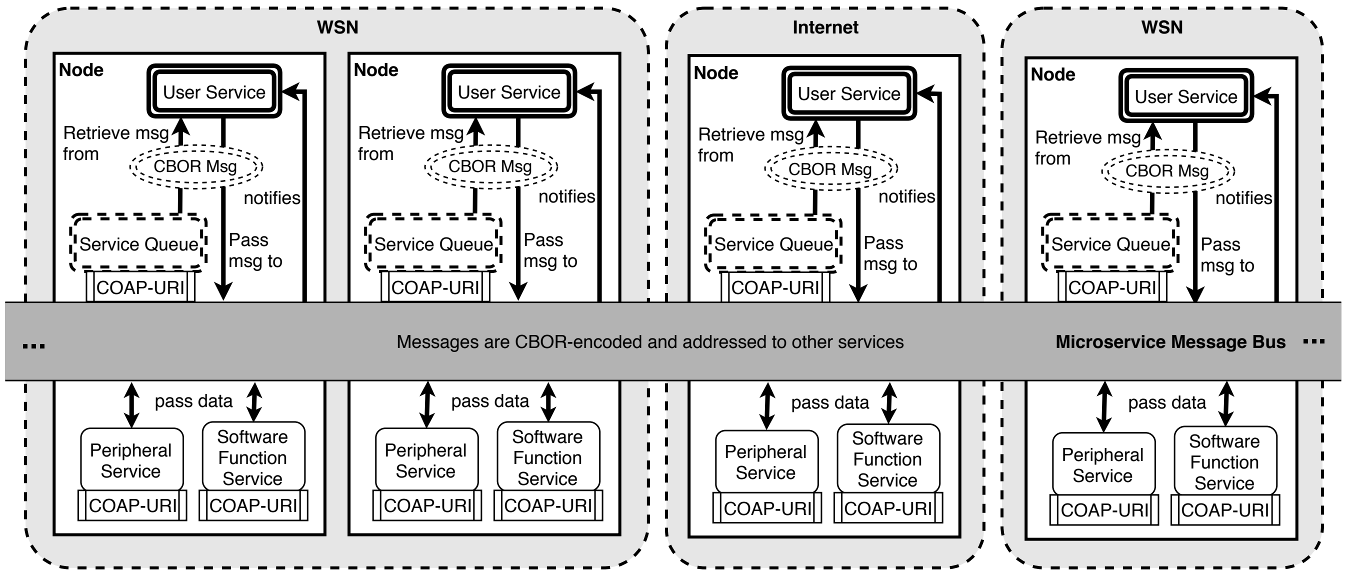


Figure 2: Overview Microservice Message Bus showing a unified communication paradigm across nodes and networks.

the message bus of the platform. The bus monitors the message and checks the content. If the message is non-local, it is first transmitted to the microservice message bus of the remote device. After, the message is either placed by the bus into the addressed service's queue, if the message was addressed to another user service, or executed as a command, if addressed to a platform service. For demonstration purposes, we implemented Java, since it is native to the IR, and C, since it is a challenging and representative example of an unsafe programming language in the IoT. Critically, languages are supported by a GLSP to build CBOR messages, making all services equally expressive by leveraging the CBOR format.

Polyglot CerberOS hosts services in multiple languages by compiling them to the IR, Java bytecode. Java is compiled with the default Java compiler, and C with LLJVM[1]. Java services and their dependencies are native to the interpreter, a Java VM, and are thus compiled and linked with the default Polyglot CerberOS runtime directly. The LLJVM tool is agnostic to Polyglot CerberOS' VM and depends on a full Java runtime, C library support and requires its own LLJVM runtime support. It compiles `.c` files to `.class` files and expects a runtime which, by default, measures in the hundreds of kilobytes, which is unsuitable to the target IETF Class-1 devices. Hence, C with LLJVM is an example of a third-party compilation which requires more stubs. For instance, `libc.java` and `Memory.java` are necessary for LLJVM to respectively provide standard library features and a managed memory space. The GLSP for C stubs `libc.java` to the default runtime and stubs `Memory.java` to native code implementations to match the memory management of LLJVM in the VM. Stubbing to the default runtime enables code reuse, stubbing to the native code allows for efficient guest language-specific implementations as necessary. Stubs avoid implementing the entire runtime and make the execution of LLJVM-compiled C services on constrained devices possible. Figure 3 provides an example for C and Java and their compilers and demonstrates how the system of monitored virtual message signatures maps down to an efficient native code implementation, solving the dependencies. After compilation, the services in IR are further optimized and loaded on the device.

[1]A set of open-source third-party tools and libraries for running low level languages on the JVM, available: https://github.com/davidar/lljvm. Last Access: September 2018.

Microservice Message Bus. The microservice message bus of Polyglot CerberOS implements two standards, CBOR [8] and COAP [25], to enable data serialisation, addressing and transparent interactions between services. Service messages are built in the CBOR format through the GLSP and addressed, following the COAP-URI scheme. Services pass messages to the bus and have a service-specific message queue on the bus they access through the GLSP. The message is delivered to local or remote services, depending on the address. For device services, the message is parsed to undertake the requested action, e.g., a temperature sensing for a peripheral. For user services, the message is put in the service's queue in the CBOR format and the service is notified. The user service can then retrieve the message from the queue and extract the data using CBOR support provided by the GLSP. Listing 1 shows how a messaging C service in the implementation constructs a message and gets an integer value through the GLSP (specified in `lib_uj_library_c.h`) even though it was compiled with an agnostic third-party tool.

```
#include "lib_uj_library_c.h"
int main(){
      // IP address of the service host.
    char dest_addr[] = "2a02::d8bc";
      // Example COAP-URI of the addressed service.
    int srvId = 5;
      // The service interface to be executed.
    char func[] = "send_packet";
      //The data to be transmitted.
    unsigned char data[] = {1,2,3,4,5,6,7,8,9,10};
      //Construct CBOR Message.
    unsigned char cbor_msg = create_cbor_msg(func, data);
      // Passing the message to the device.
    pass_msg(dest_addr, srvId, cbor_msg);
```

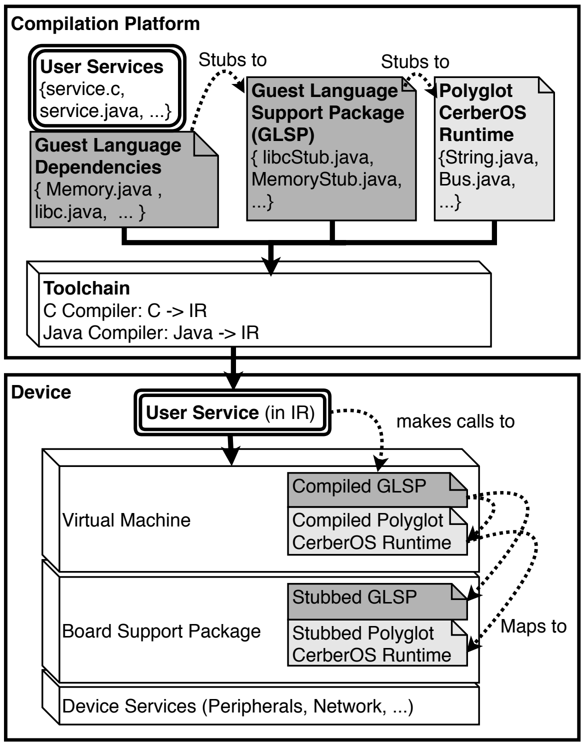

Compilation Platform
User Services {service.c, service.java, ...}
Stubs to
Stubs to
Guest Language Support Package (GLSP) { libcStub.java, MemoryStub.java, ...}
Polyglot CerberOS Runtime {String.java, Bus.java, ...}
Guest Language Dependencies { Memory.java , libc.java, ... }
Toolchain
C Compiler: C -> IR
Java Compiler: Java -> IR
Device
User Service (in IR)
makes calls to
Virtual Machine
Compiled GLSP
Compiled Polyglot CerberOS Runtime
Board Support Package
Stubbed GLSP
Stubbed Polyglot CerberOS Runtime
Maps to
Device Services (Peripherals, Network, ...)

Figure 3: Overview and example of the compilation and execution of a service in two languages.

```
        // Retrieves an int from the queue when notified.
    int value = retrieve_int_from_queue();
    return 0;
}
```

Listing 1: Example of a messaging C service.

6 EVALUATION

This section repeats the evaluation done by [2] to demonstrate Polyglot CerberOS as a resource-secure multi-tenant platform with multilingualism and enhanced interoperability. During evaluation, we compare Polyglot CerberOS implementations of sample IoT services to native implementations on plain Contiki and implementations on our previous work, henceforth called Monoglot CerberOS. Previous benchmark results for Monoglot CerberOS showed that (i) memory impact is within feasible bounds for IETF Class-1 device constraints, (ii) execution performance and battery lifetime are feasible for IoT services that do not do compute-heavy tasks and have sampling rates higher than 10 seconds and (iii) services in bytecode significantly reduce the amount of needed ROM, giving benefits in deployment energy cost and time but require more RAM at runtime. Summarized, Monoglot CerberOS achieved secure multi-tenancy for Java services on IoT platforms in non-real time embedded IoT use case scenarios.

Polyglot CerberOS is evaluated on μPnP, a representative IoT platform [27] running Contiki 3.0 [13] on the ATmega1284P board, an 8-bit microcontroller with 128KB flash, 16KB RAM and clocked at 10MHz. μPnP provides an efficient IP network stack and access to features such as timers and an I/O API. Three core benchmark services are evaluated:

Sensor The sensor service reads a sensor measurement from a peripheral, using the I2C bus.

Storage The storage service writes a 50-byte array as a block to flash memory.

Network The network service sends a 50-byte payload packet to a gateway.

These services determine how Polyglot CerberOS performs for typical IoT tasks. Note that the evaluated services are the same in function and output as the ones from [2], but differ in programming since user services now access device services (e.g., network, sensors) through a monitored message passing paradigm, as Listing 1 demonstrates.

Table 2: Memory cost in ROM and RAM in bytes of Polyglot CerberOS and the optimised LLJVM runtime compared to Monoglot CerberOS and the default LLJVM runtime.

	ROM	RAM	ΔROM	ΔRAM
Monoglot CerberOS	64458	4616	-	-
Polyglot CerberOS	80376	8642	+24.7%	+87.2%
Default C Runtime	>500KB	>2000KB	-	-
Optimised C Runtime	<10KB	<2KB RAM	-	-

6.1 Memory Analysis Polyglot CerberOS

Memory Size Firmware. Polyglot CerberOS requires additional ROM and RAM for supporting interoperability and multilingualism. At this stage of research, we have not yet focused on optimizing the firmware size but added the design alongside existing, sometimes deprecated, mechanisms. Optimisation is a target of future work. The memory size evaluation shows that Polyglot CerberOS remains reasonable despite the added overhead. Table 2 presents the analysis. Polyglot CerberOS is designed to integrate with other platforms and to be modular, where, for instance, runtime support for guest languages (i.e., the stubs) can be removed or added at flash time. Since the size of the GLSP in bytecode and native code stubs varies depending on the language and implementation we do not consider this separately but compare Polyglot CerberOS with support for Java and C as guest languages to Monoglot CerberOS. Unoptimized Polyglot CerberOS requires around 16KB ROM and 4KB RAM for the support of additional guest languages and for the implementation of the microservice message bus.

Table 2 shows the memory cost of the full C runtime for the default compilation tool and compares it to an optimized implementation containing just the essential parts. Since stubs and their implementation depend on the language and its compiling tool, they may be implemented at either the bytecode level or at the native level, with different implications on size. Here, a significant part of the ROM and RAM savings in the optimised C runtime are due to stubbing the memory management of an unsafe language at native code level. Related research [5] shows similar sizes for a VM and embedded implementations of C libraries.

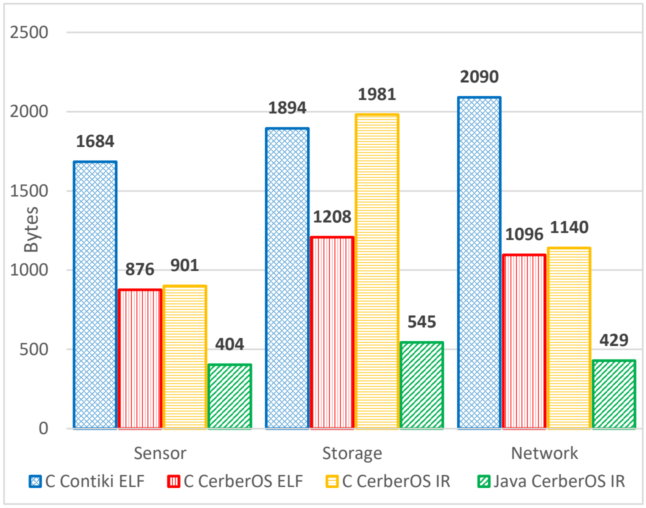
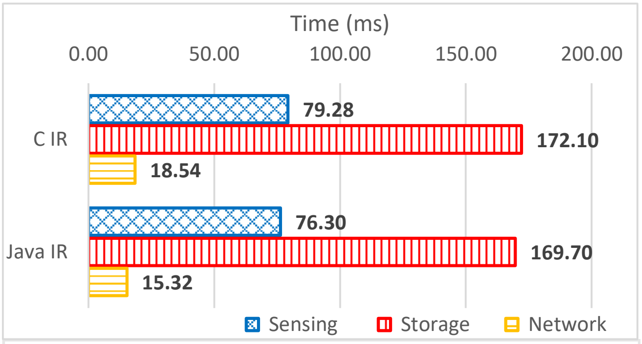


Figure 4: Service ROM size in bytes for C as Contiki ELF, CerberOS ELF and CerberOS IR and Java as CerberOS IR.

Memory Size of Services. The memory required to run a guest language service depends on two factors: (i) how the guest language is represented in the IR, which depends on the compiler, and (ii) the additional memory required for resource management per service (i.e., the resource contract and buckets).

We compare four different service formats: a C service with a native Contiki implementation in the ELF module format, a C service in a Polyglot CerberOS implementation in the ELF module format, a C service as an exemplary language compiled to the IR and a Java service as an example of efficient compilation to the IR (bytecode). Note that the Contiki implementation differs in coding paradigm; all other services are implemented through message passing, whereas the Contiki service sends native messages.

Figure 4 shows the compiled ROM sizes for the three services. The secured languages are smaller in the case of Java while the relative sizes of other services varies. The CerberOS services show the benefit of the message passing model when compared to a Contiki implementation which blows up the size due to its process system. Comparing C CerberOS ELF to C CerberOS IR shows the inefficiency of the compilation tool for some services, where verbose calls add to the overall size. This is particularly acute for the `Storage` service, where array construction in the service requires many redundant calls. Verbose calls can be improved by an optimized compiler or in post-processing of the IR. Overall, this analysis reveals Polyglot CerberOS can obtain significant size reductions for services, with the biggest gains apparent for Java services. The static RAM consumption of services is similar to Monoglot CerberOS, where a service is a thread with a minimal cost of 218 bytes for its stack. Each service also requires a non-fixed RAM cost for the contract and buckets. In practice, a service costs between 400 to 900 bytes of RAM, depending on its contract and data structures, and more for C services due to the redundant stack space requirements introduced by the compilation tool. In comparison, a simple example application in Contiki [13] required 200 bytes of RAM.



Poly/Mono Java Ratio ⍰	1.072 ⍰	1.000 ⍰	0.971 ⍰
⍰/Java IR Ratio ⍰	1.039 ⍰	1.0141 ⍰	1.210 ⍰

Figure 5: Execution time in ms of C and Java services and ratios of Monoglot and Polyglot and C and Java services.

6.2 Polyglot CerberOS Execution Performance

Execution performance of services depends on three factors: (i) the representation of the service in IR, which depends on the language and compilation tool, (ii) the overhead of resource management and (iii) the time overhead of the message passing paradigm.

Figure 5 shows the execution times of interpreted C and Java services on Polyglot CerberOS graphically and includes a table showing their ratio as well as the ratio of Java services on Monoglot CerberOS compared to Polyglot CerberOS. The Polyglot/Monoglot Java ratio reveals the effect of the message passing paradigm on execution times for Java services, directly comparing the performance of Polyglot and Monoglot CerberOS for the same type of services. The performance impact is minimal and varies according to how the Java service is programmed in Polyglot and Monoglot CerberOS. The ratio of the performance for C in IR to Java for the benchmark services compares the efficiency of the LLJVM compiler to the ideal case Java compiler in representing services in IR. For regular board access and standard operations (sending messages, giving commands, getting results), C services are between 1% and 20% less performant. A similar analysis for computation-heavy services shows significantly increased overhead (up to a factor 25) due to many verbose calls caused by the agnostic compiler tool, as discussed for the memory size of the service.

Overhead Message Passing Paradigm. Interoperability across platform and language barriers is done by passing standards-based messages locally or remotely, which requires serialisation of data and incurs overhead. We measure two types of overhead for the message passing model: (i) the time required to pass 60 bytes of data (e.g., an address and command string) to the bus using the supporting library by a C service and a Java service and (ii) the time required to encode this 60-byte data array to the CBOR format. This allows to clearly determine the execution impact.

The time for passing a 60-byte message by the C IR is 0.1675*ms*, while for Java this is 0.0909*ms*. The time required for encoding a 60-byte data array to the CBOR format is 0.1872*ms*. Compared to the results of Figure 5, the timing overhead of message passing and encoding is at least two orders of magnitude lower than overall execution time. This result is on-par with other research [17] and reveals that performing service operations through message passing and serialisation is feasible for constrained devices.

6.3 Energy Analysis of Services

Feasibility of software techniques for battery-powered IoT devices is often determined by the impact on the energy consumption. Resource security, multilingualism and interoperability are only interesting in so far that the device can maintain sufficiently long battery lifetimes. We evaluate this explicitly by repeating the device energy consumption analysis from [2].

Figure 6 presents the battery lifetime in years for node sampling rates in seconds for the three benchmark services in native Contiki C, C in IR and Java in IR for both Monoglot and Polyglot CerberOS. The figures shows that the impact on battery lifetime for the new services on Polyglot CerberOS is reasonable for both C and Java for sampling rates exceeding 10s. In addition, the difference in battery lifetime impact between native, monoglot or polyglot services is small and decreases with the sampling rate as well. The same exercise for computation-heavy services gives a feasible impact for sampling rates of up to several hours which is reasonable for constrained battery-powered devices doing periodic heavy computations. This result establishes that the main conclusion of [2] still holds: battery lifetime for IoT services on Polyglot CerberOS are feasible for services that rarely do compute-heavy tasks and have sampling rates higher than 10 seconds.

6.4 Discussion

Evaluation shows that implementing an OS based on the design outlined in Section 4 with multi-tenancy, resource-security, multilingualism and interoperability can be done within the capabilities of constrained IoT devices. Specifically, Polyglot CerberOS is feasible for IETF Class-1 devices in terms of memory, both in firmware and service sizes, and execution performance is reasonable given typical non-real time IoT use cases with common sample rates of minutes. The overhead added by Polyglot CerberOS beyond interpretation is in two areas: (i) the cost of interpretation worsens depending on how efficiently the service can be represented in the IR due to verbose calls introduced by the compilation tool and (ii) the message passing paradigm needed for interoperability introduces additional overhead since data needs to be serialised.

The cost is mainly determined by the first factor and this is directly influenced by the compilation tool and the IR. Since the used third-party tool for C-to-bytecode compilation is not optimized for CerberOS or embedded devices, significant improvements are possible. The evaluation should therefore be considered to demonstrate worst-case scenarios. Mainstream computing research such as Truffle/GraalVM [14] has shown that specific tools suited to their environment can obtain performance approaching that of native implementations.

7 CONCLUSIONS AND FUTURE WORK

This paper presented Polyglot CerberOS, the first resource-secure multilingual OS supporting interoperable heterogeneous services and multi-tenancy for IETF Class-1 IoT devices. Polyglot CerberOS facilitates multi-tenant IoT platforms by enabling services to be implemented in multiple languages while providing transparency for developers, secure interactions and resource control. IoT deployments can be safely and transparently shared between different parties since whatever programming language is used, the service can securely communicate with others and has contractual guarantees on the provision and use of key resources such as computation, storage, sensors, actuators and energy. For instance; developers can program services in C and share the same platform as developers programming in Java while obtaining the same benefits in language properties (e.g., memory safety), remaining interoperable and being assured of its contractual resource access and security.

Polyglot CerberOS builds upon our previous work, a resource-secure OS for sharing IoT devices between Java services, to support additional languages, interoperability and developer transparency. Multiple services in different programming languages run on the same device while maintaining resource security through an advanced automatic toolchain and standards-based paradigm. Services are compiled and executed in an IR on the device through a generic system of monitored virtual message signatures that are subject to access control and monitoring and mapped to an efficient native implementation through a virtual machine. Service interoperability is achieved by generalising device service access through a common data serialisation format and addressing scheme on top of a microservice message bus. Evaluation shows the implementation is efficient even on IETF Class-1 devices, still supporting multi-year battery lifetimes, without requiring special hardware or software modules. The contributions of Polyglot CerberOS are (i) ensuring isolation, resource security and secure interactions between diverse services, peripherals and platforms across multiple programming languages and (ii) secure interoperability and transparency between these entities across multiple programming languages. In summary, Polyglot CerberOS provides strong multilingual security guarantees and transparency to service developers for interoperable multi-tenant commodity IoT devices.

Since Polyglot CerberOS is a fully fledged OS, plenty of research challenges remain. Future work on Polyglot CerberOS can be divided into three categories: compilation, interpretation and interoperability. A target for improvement is tailoring the compilation tools of the supported languages so that the compiled version of the language in the IR is better optimized to the runtime environment of CerberOS. A dedicated compilation tool or post-processing of the compiled services can reduce the amount of stubs and avoid verbose calls. Interpretation and the IR are interesting targets for improvement since Polyglot CerberOS now uses standard Java bytecode with minor optimisations. A dedicated IR designed for embedded devices can unlock the benefits of Polyglot CerberOS more efficiently as long as it is interpreted. Finally, another opportunity is the interoperability mechanism, which for now only offers point-to-point message passing. Interesting possibilities are implementing subscription modules such as multicast/anycast on the message bus and (soft) real-time message guarantees.

ACKNOWLEDGMENTS

This research is partially funded by the Research Fund KU Leuven and the FWO D3-CPS project.

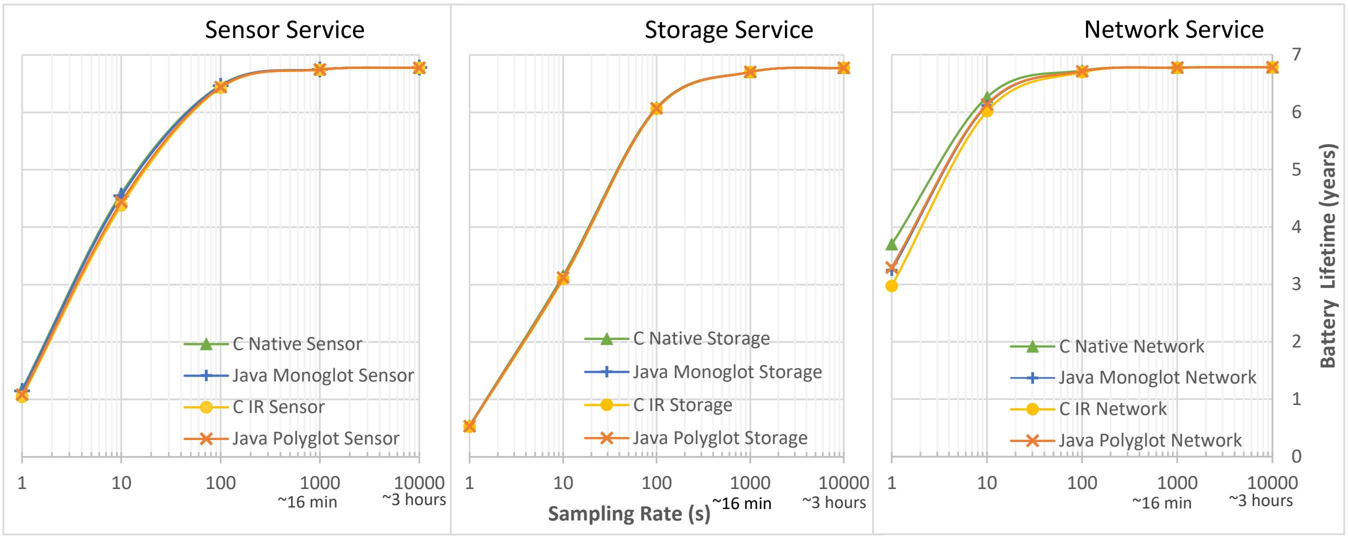


Figure 6: Battery lifetime in years for node sampling rates in seconds on a log scale for the benchmark services.

REFERENCES

[1] Anthony A. Aaby. 1996. Introduction to Programming Languages. (1996), 1495.

[2] Sven Akkermans, Wilfried Daniels, Gowri Sankar R., Bruno Crispo, and Danny Hughes. 2017. CerberOS: A Resource-Secure OS for Sharing IoT Devices. In *Proceedings of the 2017 International Conference on Embedded Wireless Systems and Networks (EWSN &#8217;17)*. Junction Publishing, USA, 96–107. http://dl.acm.org/citation.cfm?id=3108009.3108023

[3] Ala Al-Fuqaha, Mohsen Guizani, Mehdi Mohammadi, Mohammed Aledhari, and Moussa Ayyash. 2015. Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. *IEEE Communications Surveys & Tutorials* 17, 4 (2015), 2347–2376. https://doi.org/10.1109/COMST.2015.2444095

[4] Emmanuel Baccelli, Oliver Hahm, Mesut Gunes, Matthias Wahlisch, and Thomas Schmidt. 2013. RIOT OS: Towards an OS for the Internet of Things. IEEE, 79–80. https://doi.org/10.1109/INFCOMW.2013.6970748

[5] Thomas W. Barr, Rebecca Smith, and Scott Rixner. 2012. Design and Implementation of an Embedded Python Run-time System. In *Proceedings of the 2012 USENIX Conference on Annual Technical Conference (USENIX ATC'12)*. USENIX Association, Berkeley, CA, USA, 27–27. http://dl.acm.org/citation.cfm?id=2342821.2342848

[6] Philip A. Bernstein. 1996. Middleware: a model for distributed system services. *Commun. ACM* 39, 2 (Feb. 1996), 86–98. https://doi.org/10.1145/230798.230809

[7] C. Bormann, M. Ersue, and A. Keranen. 2014. *Terminology for Constrained-Node Networks*. RFC 7228. RFC Editor. http://www.rfc-editor.org/rfc/rfc7228.txt

[8] C. Bormann and P. Hoffman. 2013. *Concise Binary Object Representation (CBOR)*. Technical Report RFC7049. RFC Editor. https://doi.org/10.17487/rfc7049

[9] Thorsten Brunklaus and Leif Kornstaedt. 2002. *A Virtual Machine for Multi-Language Execution*. Technical Report. Programming Systems Lab, Universitat des Saarlandes, Saarbrucken, Germany. 10 pages.

[10] A. Caracas, T. Kramp, M. Baentsch, M. Oestreicher, T. Eirich, and I. Romanov. 2009. Mote Runner: A Multi-language Virtual Machine for Small Embedded Devices. IEEE, 117–125. https://doi.org/10.1109/SENSORCOMM.2009.27

[11] Henry Chesbrough. 2017. The Future of Open Innovation: The future of open innovation is more extensive, more collaborative, and more engaged with a wider variety of participants. *Research-Technology Management* 60, 1 (Jan. 2017), 35–38. https://doi.org/10.1080/08956308.2017.1255054

[12] Wilfried Daniels, Danny Hughes, Mahmoud Ammar, Bruno Crispo, Nelson Matthys, and Wouter Joosen. 2017. SμV - the security microvisor: a virtualisation-based security middleware for the internet of things. ACM Press, 36–42. https://doi.org/10.1145/3154448.3154454

[13] A. Dunkels, B. Gronvall, and T. Voigt. 2004. Contiki - a lightweight and flexible operating system for tiny networked sensors. IEEE (Comput. Soc.), 455–462. https://doi.org/10.1109/LCN.2004.38

[14] Matthias Grimmer, Chris Seaton, Roland Schatz, Thomas Wurthinger, and Hanspeter Mossenbock. 2015. High-performance cross-language interoperability in a multi-language runtime. ACM Press, 78–90. https://doi.org/10.1145/2816707.2816714

[15] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. 2013. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems* 29, 7 (Sept. 2013), 1645–1660. https://doi.org/10.1016/j.future.2013.01.010

[16] Danny Hughes, Klaas Thoelen, Wouter Horr'e, Nelson Matthys, Javier Del Cid, Sam Michiels, Christophe Huygens, and Wouter Joosen. 2009. LooCI: a loosely-coupled component infrastructure for networked embedded systems. ACM Press, 195. https://doi.org/10.1145/1821748.1821787

[17] Pavel Kalvoda. 2014. Implementation and evaluation of the CBOR protocol. , 116 pages.

[18] Niklas Kolbe, Jérémy Robert, Sylvain Kubler, and Yves Le Traon. 2017. PROFICIENT: Productivity Tool for Semantic Interoperability in an Open IoT Ecosystem. In *Proceedings of the 14th International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*. 10.

[19] In Lee and Kyoochun Lee. 2015. The Internet of Things (IoT): Applications, investments, and challenges for enterprises. *Business Horizons* 58, 4 (July 2015), 431–440. https://doi.org/10.1016/j.bushor.2015.03.008

[20] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler. 2005. TinyOS: An Operating System for Sensor Networks. In *Ambient Intelligence*, Werner Weber, Jan M. Rabaey, and Emile Aarts (Eds.). Springer-Verlag, Berlin/Heidelberg, 115–148. https://doi.org/10.1007/3-540-27139-2_7

[21] Anne H. H. Ngu, Mario Gutierrez, Vangelis Metsis, Surya Nepal, and Michael Z. Sheng. 2016. IoT Middleware: A Survey on Issues and Enabling technologies. *IEEE Internet of Things Journal* (2016), 1–1. https://doi.org/10.1109/JIOT.2016.2615180

[22] Brian Oki, Manfred Pfluegl, Alex Siegel, and Dale Skeen. 1994. The Information Bus: an architecture for extensible distributed systems. In *ACM SIGOPS Operating Systems Review*, Vol. 27. ACM, 58–68.

[23] Krassimira Antonova Paskaleva. 2011. The smart city: A nexus for open innovation? *Intelligent Buildings International* 3, 3 (July 2011), 153–171. https://doi.org/10.1080/17508975.2011.586672

[24] Manuel Rigger, Roland Schatz, Matthias Grimmer, and Hanspeter Mossenbock. 2017. Lenient Execution of C on a Java Virtual Machine: or: How I Learned to Stop Worrying and Run the Code. ACM Press, 35–47. https://doi.org/10.1145/3132190.3132204

[25] Z. Shelby, K. Hartke, and C. Bormann. 2014. *The Constrained Application Protocol (CoAP)*. Technical Report RFC7252. RFC Editor. https://doi.org/10.17487/rfc7252

[26] M. U.Farooq, Muhammad Waseem, Anjum Khairi, and Sadia Mazhar. 2015. A Critical Analysis on the Security Concerns of Internet of Things (IoT). *International Journal of Computer Applications* 111, 7 (Feb. 2015), 1–6. https://doi.org/10.5120/19547-1280

[27] Fan Yang, Nelson Matthys, Rafael Bachiller, Sam Michiels, Wouter Joosen, and Danny Hughes. 2015. μPnP: plug and play peripherals for the internet of things. ACM Press, 1–14. https://doi.org/10.1145/2741948.2741980

[28] Ibrar Yaqoob, Ejaz Ahmed, Ibrahim Abaker Targio Hashem, Abdelmuttlib Ibrahim Abdalla Ahmed, Abdullah Gani, Muhammad Imran, and Mohsen Guizani. 2017. Internet of Things Architecture: Recent Advances, Taxonomy, Requirements, and Open Challenges. *IEEE Wireless Communications* 24, 3 (June 2017), 10–16. https://doi.org/10.1109/MWC.2017.1600421